Mise en œuvre d'un solveur direct parallèle pour l'inversion des problèmes locaux au sein d'une méthode de décomposition de domaine

Ibrahima Gueye, Xavier Juvigny, Frédéric Feyel, François-Xavier Roux, Georges Cailletaud

Mise en œuvre d'un solveur direct parallèle pour l'inversion des problèmes locaux au sein d'une méthode de décomposition de domaine

I. GUEYE[1,2], X. JUVIGNY[1], F. FEYEL[1], F.-X. ROUX[1], G. CAILLETAUD[2]

[1] ONERA - Centre de Châtillon
29, Avenue de la Division Leclerc 92322 Châtillon Cedex.
{ibrahima.gueye,frederic.feyel}@onera.fr

[2] Centre des Matériaux - Mines ParisTech - UMR CNRS 7633
B.P. 87 91003 EVRY Cedex.
georges.cailletaud@ensmp.fr

Résumé — Le but de ce papier est de mettre au point un algorithme parallèle pour la résolution directe de grands systèmes linéaires creux et l'intégrer dans les méthodes de décomposition de domaine. Ces systèmes linéaires, souvent rencontrés lor de la simulation numérique de problèmes de mécanique des structures par des codes de calcul par éléments finis, sont résolus avec des coûts très importants en temps ce calcul et en espace mémoire. Dans ce papier, un parallélisme à deux niveaux a été exploité. L'exploitation du niveau inférieur de parallélisme a d'abord consisté à réaliser un solveur direct parallèle basé sur une technique de dissection emboitée et l'intégrer ensuite dans les méthodes FETI. Ce solveur direct a l'avantage de traiter automatiquement et proprement les modes à énergie nulle dans des structures flottantes. L'exploitation du niveau supérieur a consisté à améliorer la phase itérative de la méthode FETI classique. Des tests ont été effectués pour évaluer les performances du solveur mis en place.

Mots clés — Parallélisme à deux niveaux, dissection emboîtée, modes à énergie nulle.

1 Introduction

La résolution directe de systèmes linéaires a toujours joué un role important dans la simulation numérique de nombreux problèmes de calcul scientifique. En calcul de structures par exemple, l'utilisation des codes de calcul par éléments finis conduit souvent à des systèmes linéaires creux de très grande taille. Leur résolution mène à des coûts importants en temps de calcul CPU et en espace mémoire.

La parallélisation de ces codes est alors une technique nécessaire. Elle est capable de réduire les coûts de calcul et par la suite, elle permet de simuler en calcul des structures des problèmes avec des lois de comportement de plus en plus complexes et des géométries affinées. Les méthodes de décomposition de domaine sont un moyen naturel permettant de paralléliser ces codes. Une des plus utilisées est la méthode FETI [1] [3]. Elle repose sur une "approche duale" consistant à introduire des conditions de raccord en effort aux interfaces entre sous-domaines. Elle présente aussi l'avantage d'être robuste et bien adaptée aux problèmes étudiés en calcul de structures. Cependant,

[1] Finite Element Tearing and Interconnecting

des études faites sur de gros modèles (dizaines de millions d'inconnues) ont montré que l'efficacité de cette méthode se détériore au delà de quelques centaines de sous-domaines. En voulant découper ces gros modèles en un nombre raisonnable de sous-domaines, un autre problème s'impose. La taille des systèmes locaux devient très importante. De plus, avec l'évolution de la technologie des microprocesseurs, on voit naître de nouvelles architectures massivement multi-cœurs. L'intérêt essentiel et la force des solutions multi-cœurs est de permettre l'exécution simultanée d'une tâche par cœur. Certains algorithmes massivement parallèles peuvent tirer pleinement profit de ces nouvelles architectures.

C'est dans ce contexte que cette étude est effectuée. Ils consistent à mettre au point un algorithme parallèle pour la résolution directe de grands systèmes linéaires creux et l'intégrer dans les méthodes de décomposition de domaine. Un parallélisme à deux niveaux est exploité. L'exploitation du plus bas niveau de parallélisme est basée dans un premier temps, à la mise en œuvre d'un solveur direct parallèle. Pour cela, nous proposons d'utiliser la technique de dissection emboîtée pour trouver un ordre d'élimination des inconnues dans l'inversion des systèmes creux [1], [5], [6]. Cette technique est bien adaptée au parallélisme multi-tâches car elle permet de diviser la factorisation du système en autant d'étapes que de niveaux dans l'algorithme de dissection. Dans un deuxième temps, le solveur direct parallèle est introduit dans les méthodes FETI [4], [7]. Ce solveur direct, basé sur la technologie du multi-threading, permet ainsi de faire de la résolution locale parallèle dans FETI et traiter automatiquement et proprement les modes à énergie nulle dans les sous-structures flottantes. Le plus haut niveau est un parallélisme à gros grains entre les sous-domaines de FETI. A ce niveau, nous améliorons la méthode itérative qui a été utilisée pour la résolution du problème d'interface de FETI. Nous évaluons les performances du solveur direct et nous le comparons à DSCPack[2] [9] qui est un solveur linéaire de référence. Pour cela, de gros calculs sont simulés avec ZéBuLon qui est un code de calcul par éléments finis pour des problèmes de mécanique non linéaires. Ce code, développé conjointement par l'ONERA, l'ENSMP et la société NW Numerics, a été parallélisé avec une méthode FETI.

Le reste du papier est structuré comme suit. Dans la prochaine section, nous décrivons les phases de mise en œuvre du solveur direct parallèle. Pour cela, nous présentons d'abord la méthode de dissection emboîtée qui a été proposée pour renuméroter les inconnues du système à résoudre. Nous montrons ensuite comment cette technique est utilisée dans les phases numérique et de résolution des systèmes triangulaires. En cas de présence de sous-structures flottantes, nous apportons un traitement pour les modes à énergie nulle. Dans la section 3, nous évaluons et comparons les performances du solveur direct avec d'autres existants dans ZéBuLon. Nous concluons dans la section 4 sur les futures étapes de ces travaux.

2 Mise en œuvre du solveur direct parallèle

Dans cette section, nous mettons en œuvre un solveur direct parallèle pour résoudre de grands systèmes linéaires $Mx = b$, M étant creuse et pouvant être symétrique ou non symétrique. La résolution de ces systèmes creux par méthode directe comprend trois grandes phases :

- la phase d'analyse qui consiste à trouver un ordre d'élimination des inconnues sous la forme d'un super-arbre et factoriser symboliquement la matrice du système M. Les techniques de renumérotation permettent d'effectuer cela. Ici, un algorithme de dissection emboîtée est utilisé. Le choix d'utiliser essentiellement cette méthode provient du fait qu'elle permet d'une part, plus de parallélisme que les algorithmes de degré minimum [10] et d'autre part, elle conduit souvent à de meilleurs résultats.
- la phase numérique détermine implicitement les matrices triangulaires inférieure et supérieure de la matrice en s'appuyant sur le super-arbre d'élimination qui a été produit ;

[2] Domain Separator Codes Package

– la troisième phase est la descente-remontée. Ici, la solution numérique du système est obtenue en résolvant les systèmes triangulaires résultants de la phase numérique.

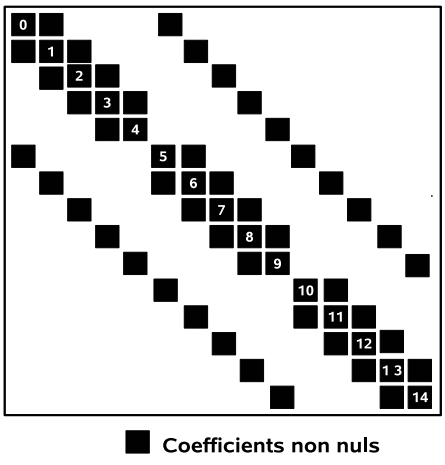
2.1 Dissection emboîtée

La dissection emboîtée est l'une des méthodes les plus attractives pour trouver un ordre d'élimination en resolvant les systèmes linéaires creux $Mx = b$. Cette méthode ordonne, avec l'aide de séparateurs, les sommets du graphe (ou maillage) associé à la matrice M du système. Le graphe $G(V,E)$ d'une matrice M de dimension n est constitué d'un ensemble V de n sommets et d'un ensemble E d'arêtes reliant les sommets de V. Chaque sommet du graphe représente une inconnue du système linéaire. Et un séparateur est un petit ensemble de sommets qui découpe le reste du graphe en deux composants disconnectés. Différents niveaux de séparateurs sont utilisés pour réordonner les sommets du graphe.

Le principe de la dissection emboîtée est basé sur la bissection récursive du graphe de la matrice M à factoriser. Une première bissection est effectuée en sélectionnant un ensemble de sommets formant un séparateur S_Γ. Ce séparateur est retiré du graphe et, cela génére une partition en deux sous-structures disconnectées I_1 et I_2. Le séparateur est choisi de telle sorte que sa taille est aussi petite possible et que les tailles des sous-structures obtenues soient équivalentes. Chacun de ces sous-graphes est alors découpé récursivement suivant le même processus, jusqu'à ce que les sous-structures générées soient de tailles suffisamment petites.

Pour illustrer le principe de cette technique, nous considérons un système linéaire creux $Mx = b$, où la structure de M est présentée sur la figure 1(a). Le graphe (ou maillage) représentant la matrice est montré sur la figure 1(b).

(a) Structure de la matrice M.



(b) Graphe ou maillage.

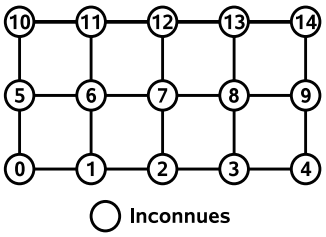
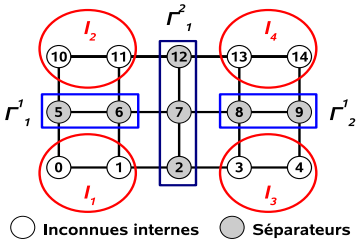


Figure 1 – Exemple de système linéaire creux.

Nous supposons que nous avons déjà effectué, avec l'aide du partitionneur de graphe METIS [8], une bissection récursive des nœuds du maillage en quatre sous-structures (voir Figure 2(a)). L'approche de la dissection emboîtée consiste à renuméroter les inconnues du système par paquets en commençant par les sous-structures I_i et en terminant par les séparateurs du graphe Γ_j^l. Les blocs diagonaux $M_{I_iI_i}$ situés au niveau 0 correspondent aux matrices associées aux sous-structures I_1, I_2, I_3 et I_4. Les blocs diagonaux $M_{\Gamma_j^l\Gamma_j^l}$ situés aux autres niveaux l correspondent aux matrices associées aux séparateurs Γ_j^l. Les blocs extra-diagonaux $M_{I_i\Gamma_j^l}$ correspondent aux connections entre les inconnues dans I_i et celles appartenant à Γ_j^l ; les blocs extra-diagonaux $M_{\Gamma_i^l\Gamma_j^m}$ $(l \neq m)$ correspondent aux connections entre les inconnues dans Γ_i^l et celles dans Γ_j^m (voir Figure 2(b)).

(a) Découpage par bissection récursive.



(b) Matrice M renumérotée.

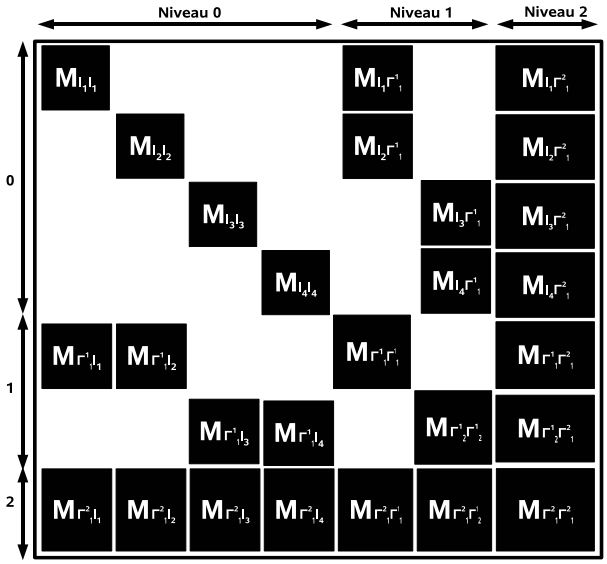


Figure 2 – Exemple d'illustration de la dissection emboitée.

Avec la stratégie de "division et conquête" de la méthode de dissection emboîtée, la parallélisation du solveur se fait naturellement. Elle génére de super-arbres d'élimination bien équilibrés dont chaque super-nœud est formé par un ensemble d'inconnues (Figure 3). Le fait que ces super-arbres soient bien équilibrés permet d'assurer un certain équilibrage des tâches. Ces arbres sont alors utilisés pour mener à bien les phases de mise en œuvre qui succédent à celle-ci.

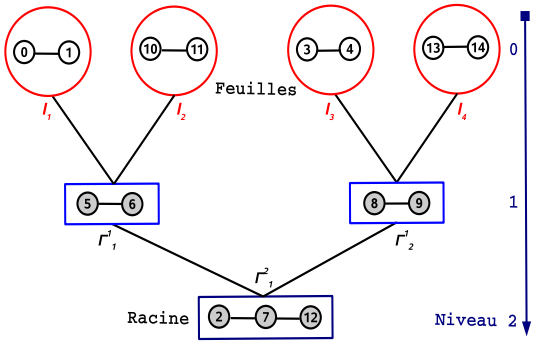


Figure 3 – Super-arbre d'élimination

2.2 Phase numérique

Une importante étape pour implanter le solveur direct est la phase numérique. Dans cette phase, nous alternons (niveau par niveau) l'étape de factorisation numérique et celle de condensation statique du système linéaire $Mx = b$ sur les inconnues appartenant aux séparateurs Γ^l_j.

2.2.1 Factorisation numérique

La factorisation numérique consiste à calculer implicitement la matrice triangulaire inférieure L et la matrice triangulaire supérieure U de M en s'appuyant sur le super-arbre généré dans la phase d'analyse. Pour cela, nous effectuons d'abord une factorisation creuse de $M_{I_iI_i}$ pour chaque sous-structure I_i sous la forme $L_{I_iI_i}U_{I_iI_i}$. Ensuite, nous faisons une factorisation pleine sous la forme

$L_{\Gamma_j^l\Gamma_j^l} U_{\Gamma_j^l\Gamma_j^l}$ de chaque bloc $S_{\Gamma_j^l\Gamma_j^l}$ de Γ_j^l ayant été créé au niveau $l-1$ de l'étape de condensation statique. Les dimensions de $M_{I_iI_i}$ et $S_{\Gamma_j^l\Gamma_j^l}$ sont souvent petites. Un solveur skyline ou frontal est utilisé pour les factoriser efficacement. Durant la factorisation, l'élimination des inconnues de chaque nœud de l'arbre peut être effectuée dès que, celles de ses fils ont été traitées.

Dans l'exemple de la section 2.1, les matrices triangulaires L et U peuvent être décrites sous une forme similaire à celle de M (voir Figure 2(b)) :

$$L = \begin{pmatrix} M_{I_1I_1} & & & & & & \\ 0 & M_{I_2I_2} & & & & & \\ 0 & 0 & M_{I_3I_3} & & & & \\ 0 & 0 & 0 & M_{I_4I_4} & & & \\ M_{\Gamma_1^1I_1} & M_{\Gamma_1^1I_2} & 0 & 0 & S_{\Gamma_1^1\Gamma_1^1} & & \\ 0 & 0 & M_{\Gamma_2^1I_3} & M_{\Gamma_2^1I_4} & 0 & S_{\Gamma_2^1\Gamma_2^1} & \\ M_{\Gamma_1^2I_1} & M_{\Gamma_1^2I_2} & M_{\Gamma_1^2I_3} & M_{\Gamma_1^2I_4} & S_{\Gamma_1^2\Gamma_1^1} & S_{\Gamma_1^2\Gamma_2^1} & S_{\Gamma_1^2\Gamma_1^2} \end{pmatrix}$$

et

$$U = \begin{pmatrix} I & 0 & 0 & 0 & M_{I_1I_1}^{-1}M_{I_1\Gamma_1^1} & 0 & M_{I_1I_1}^{-1}M_{I_1\Gamma_1^2} \\ & I & 0 & 0 & M_{I_2I_2}^{-1}M_{I_2\Gamma_1^1} & 0 & M_{I_2I_2}^{-1}M_{I_2\Gamma_1^2} \\ & & I & 0 & 0 & M_{I_3I_3}^{-1}M_{I_3\Gamma_2^1} & M_{I_3I_3}^{-1}M_{I_3\Gamma_1^2} \\ & & & I & 0 & M_{I_4I_4}^{-1}M_{I_4\Gamma_1^2} & M_{I_4I_4}^{-1}M_{I_4\Gamma_1^2} \\ & & & & I & 0 & S_{\Gamma_1^1\Gamma_1^1}^{-1}S_{\Gamma_1^1\Gamma_1^2} \\ & & & & & I & S_{\Gamma_2^1\Gamma_2^1}^{-1}S_{\Gamma_2^1\Gamma_1^2} \\ & & & & & & I \end{pmatrix}$$

2.2.2 Condensation statique

L'idée de la condensation statique est d'inverser un système local pour chaque nœud du super-arbre. Le complément de Schur global S défini par l'équation (1) est ainsi assemblé implicitement niveau par niveau.

$$S = \begin{pmatrix} S_{\Gamma_1^1\Gamma_1^1} & 0 & S_{\Gamma_1^1\Gamma_1^2} \\ 0 & S_{\Gamma_2^1\Gamma_2^1} & S_{\Gamma_2^1\Gamma_1^2} \\ S_{\Gamma_1^2\Gamma_1^1} & S_{\Gamma_1^2\Gamma_2^1} & S_{\Gamma_1^2\Gamma_1^2} \end{pmatrix} \tag{1}$$

Nous commençons d'abord la condensation statique en éliminant les inconnues dans chaque sous-structure I_i. L'élimination de ces inconnues équivaut au calcul des compléments Schur locaux ou contributions $M_{\Gamma_j^lI_i}{M_{I_iI_i}}^{-1}M_{I_i\Gamma_k^m}$ de chaque sous-structure I_i. Ces contributions, déterminées efficacement avec l'aide de la librairie BLAS de niveau 3 [2], permettent de calculer les blocs $S_{\Gamma_j^l\Gamma_k^m}$. Le calcul des blocs situés au niveau suivant ($l=1$ et $m=1$) est alors effectué complétement et celui des blocs situés aux niveaux plus élevés ($l>1$ et $m>1$) est effectué partiellement. La condensation statique se poursuit ensuite jusqu'à la racine de l'arbre. Nous éliminons les inconnues de chaque nœud Γ_j^l situé à un niveau $l>0$ et nous calculons ses contributions $S_{\Gamma_k^m\Gamma_j^l}{S_{\Gamma_j^l\Gamma_j^l}}^{-1}S_{\Gamma_j^l\Gamma_p^n}$ pour les blocs $S_{\Gamma_k^m\Gamma_p^n}$ localisés aux niveaux supérieurs m et n.

2.3 Résolution des systèmes triangulaires

La solution du système linéaire creux $Mx=b$ est déterminée en résolvant les systèmes triangulaires $Ly=b$ et $Ux=y$. Les matrices triangulaires L et U sont issues de la factorisation numérique de M. La résolution de ces systèmes est réalisée en trois étapes.

– Une étape de descente qui consiste à résoudre les systèmes locaux $(L_{I_iI_i}U_{I_iI_i})y_{I_i} = b_{I_i}$ et mettre à jour les termes $b_{\Gamma^l_j}$ du second membre b :

$$\begin{pmatrix} b_{\Gamma^1_1} \\ b_{\Gamma^1_2} \\ b_{\Gamma^2_1} \end{pmatrix} = \begin{pmatrix} b_{\Gamma^1_1} \\ b_{\Gamma^1_2} \\ b_{\Gamma^2_1} \end{pmatrix} - \begin{pmatrix} M_{\Gamma^1_1 I_1} y_{I_1} + M_{\Gamma^1_1 I_2} y_{I_2} \\ M_{\Gamma^1_2 I_3} y_{I_3} + M_{\Gamma^1_2 I_4} y_{I_4} \\ \sum_{i=1}^{4} M_{\Gamma^2_1 I_i} y_{I_i} \end{pmatrix}.$$

– Une étape qui résoût le problème condensé aux séparateurs. Cette étape est effectuée niveau par niveau dans l'arbre :

$$\begin{pmatrix} S_{\Gamma^1_1\Gamma^1_1} & 0 & 0 \\ 0 & S_{\Gamma^1_2\Gamma^1_2} & 0 \\ S_{\Gamma^2_1\Gamma^1_1} & S_{\Gamma^2_1\Gamma^1_2} & S_{\Gamma^2_1\Gamma^2_1} \end{pmatrix} \begin{pmatrix} I & 0 & S^{-1}_{\Gamma^1_1\Gamma^1_1} S_{\Gamma^1_1\Gamma^2_1} \\ 0 & I & S^{-1}_{\Gamma^1_2\Gamma^1_2} S_{\Gamma^1_2\Gamma^2_1} \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} x_{\Gamma^1_1} \\ x_{\Gamma^1_2} \\ x_{\Gamma^2_1} \end{pmatrix} = \begin{pmatrix} b_{\Gamma^1_1} \\ b_{\Gamma^1_2} \\ b_{\Gamma^2_1} \end{pmatrix}.$$

– Une étape de remontée qui calcule les solutions locales x_{I_i} :

$$\begin{pmatrix} x_{I_1} \\ x_{I_2} \\ x_{I_3} \\ x_{I_4} \end{pmatrix} = \begin{pmatrix} y_{I_1} \\ y_{I_2} \\ y_{I_3} \\ y_{I_4} \end{pmatrix} - \begin{pmatrix} M^{-1}_{I_1I_1} M_{I_1\Gamma^1_1} x_{\Gamma^1_1} + M^{-1}_{I_1I_1} M_{I_1\Gamma^2_1} x_{\Gamma^2_1} \\ M^{-1}_{I_2I_2} M_{I_2\Gamma^1_1} x_{\Gamma^1_1} + M^{-1}_{I_2I_2} M_{I_2\Gamma^2_1} x_{\Gamma^2_1} \\ M^{-1}_{I_3I_3} M_{I_3\Gamma^1_2} x_{\Gamma^1_2} + M^{-1}_{I_3I_3} M_{I_3\Gamma^2_1} x_{\Gamma^2_1} \\ M^{-1}_{I_4I_4} M_{I_4\Gamma^1_2} x_{\Gamma^1_2} + M^{-1}_{I_4I_4} M_{I_4\Gamma^2_1} x_{\Gamma^2_1} \end{pmatrix}.$$

2.4 Prise en compte des systèmes singuliers

Il existe beaucoup solveurs directs séquentiels ou parallèles mais, peu prennent en compte les singularités des systèmes non inversibles. Ces singularités sont souvent d'origine physique ou géométrique ou dûes au découpage en sous-domaines. Notre but est d'avoir un solveur qui traite automatiquement et proprement les modes à énergie nulle de ces systèmes.

Le traitement consiste à détecter, dans un premier temps, les singularités locales. Pour cela, nous cherchons des pivots proches de zéro durant la factorisation des matrices $M_{I_iI_i}$ et $S_{\Gamma^l_j\Gamma^l_j}$. Si par exemple, nous rencontrons un pivot proche de zéro à une k^{e} position de $M_{I_iI_i}$, nous renvoyons le traitement de l'inconnue correspondante à la fin de la factorisation de M. Dans un deuxième temps coincidant avec la fin de la factorisation, nous recherchons les singularités globales du système. Nous condensons d'abord le système sur les singularités détectées pour obtenir une petite matrice pleine S_s. Ensuite, nous effectuons une élimination de Gauss avec pivotage sur cette matrice. Les pivots nuls rencontrés lors de cette élimination sont alors les singularités globales. Elles sont donc utilisées pour calculer les modes à énergie nulle constituant la base du noyau de M.

2.5 Parallélisation du solveur

La stratégie de "division et conquête" de la dissection emboîtée conduit à un haut niveau de parallélisme. C'est pour cela que la parallélisation du solveur se fait naturellement. Étant donné qu'il n'y a pas de dépendances entre les super-nœuds situés à un même niveau de l'arbre d'élimination, nous traitons ces super-nœuds simultanément sur différents cœurs. L'approche qui est utilisée pour paralléliser le solveur repose sur l'utilisation de la méthodologie du multi-threading. Celle-ci, basée sur du multi-tâches et fonctionnant sur des machines à mémoire partagée, peut permettre au solveur de tirer profit des nouvelles machines multi-cœurs. Les threads POSIX (Pthreads) et une version OpenMP de la librairie MKL d'Intel sont utilisés pour implanter des threads dans les grandes phases de mise en œuvre du solveur. Une parallélisation hybride (Pthreads + OpenMP) est aussi effectuée dans la phase de factorisation numérique.

3 Évaluation des performances

Les performances d'un solveur direct peuvent être évaluées de différentes manières. Pour le solveur mis en place, nous utilisons les temps CPU et Elapsed nécessaires pour effectuer la phase d'analyse, la phase numérique et la résolution des systèmes triangulaires. Le solveur étant déjà implanté dans le code ZéBuLon, nous allons d'abord comparer les performances en séquentiel du solveur avec d'autres disponibles dans ce code. Nous effectuons ensuite une analyse de ses performances en parallèle.

Les tests d'évaluation portent sur des problèmes d'élasticité linéaire en 3D. Ces tests sont effectués, en séquentiel sur une machine bi-processeurs Intel Quad-Core Xeon 64-bit, avec une RAM de 32 Go et une fréquence de 3.16 GHz ; en paralléle sur un cluster ayant 51 nœuds qui sont des bi-processeurs AMD Opteron 64-bit possédant une RAM allant de 2 à 16 Go.

3.1 Analyse des performances en séquentiel

Nous comparons d'abord les temps d'exécution CPU du solveur Dissection qui a été développé avec ceux d'autres solveurs existants dans ZéBuLon. Les résultat sont présentés sur le tableau 1. DSCPack est un solveur direct basé sur une approche similaire à celle de la dissection emboîtée mais il ne prend pas en compte les systèmes singuliers. Par contre, Sparse Direct et Frontal sont deux autres solveurs directs qui eux tiennent compte des singularités de ces systèmes.

Solveurs \ Tailles des problèmes	107811	206763	397953
DSCPack	45.41	154.2	557.8
Dissection	67.17	207.9	776.2
Sparse Direct	871.7	3292	+10440
Frontal	2185	9972	***

Tableau 1 – Temps d'exécution CPU (s).

Sur les résultats du tableau 1, nous observons que les performances de DSCPack sont un peu meilleures que celles de notre solveur Dissection. Mais, on ne doit pas rougir de cela puisque DSCPack n'est pas capable de traiter les modes à énergie nulles des sous-structures flottantes. Et c'est là que le solveur que nous avons mis en place devient alors plus profitable pour les méthodes FETI que les autres solveurs existants dans ZéBuLon.

Le solveur Dissection peut être utilisé pour résoudre des systèmes linéaires à seconds membres multiples. Sur le tableau 2, nous montrons les performances de cette phase de descente-remontée du solveur pour un problème contenant 206763 degrés de liberté.

Solveurs \ Nombre de rhs	1	50	100	150	200
DSCPack	1.246	62.32	124.6	186.9	249.3
Dissection	2.627	58.47	114.9	172.7	231.1

Tableau 2 – Temps elapsed de descente-remontée (s).

L'analyse des résultats sur le tableau 2 nous permet de voir que nous avons de bonnes performances avec le nouveau solveur Dissection dans la phase de descente-remontée. Cela pourrait être d'un grand intérêt pour l'amélioration de la résolution du problème d'interface de FETI où

plusieures descente-remontées sont effectuées successivement pour satisfaire la continuité aux interfaces entre sous-domaines.

3.2 Analyse des performances en parallèle

Nous analysons les performances obtenues pour résoudre un grand problème d'élasticité linéaire ayant 397953 degrés de liberté. Dans la figure 4, nous présentons les temps d'exécution pour les versions parallèles du solveur mises en œuvre. Ces versions parallèles sont :

- une version multi-threads avec des threads POSIX ;
- une version utilisant version OpenMP de la librairie MKL d'Intel ;
- une version hybride avec des threads POSIX et OpenMP.

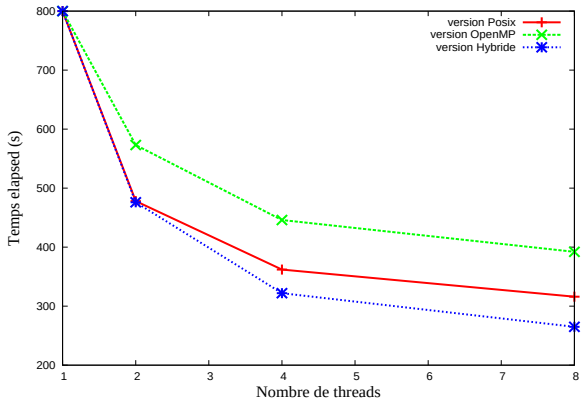


Figure 4 – Temps d'exécution (s) des 3 versions multi-threads.

Pour ce problème, nous atteignons un gain optimal en temps d'exécution avec un facteur d'environ 2,5 quand le nombre de threads est égal à 4.

Le solveur Dissection a été introduit dans FETI et il est actuellement utilisé comme solveur local. Quelques tests ont été effectués en parallèle. La taille globale de ces tests croît linéairement avec le nombre de sous-domaines (voir figure 5(a)).

(a) Problème global : tailles. (b) Taille par sous-domaine.

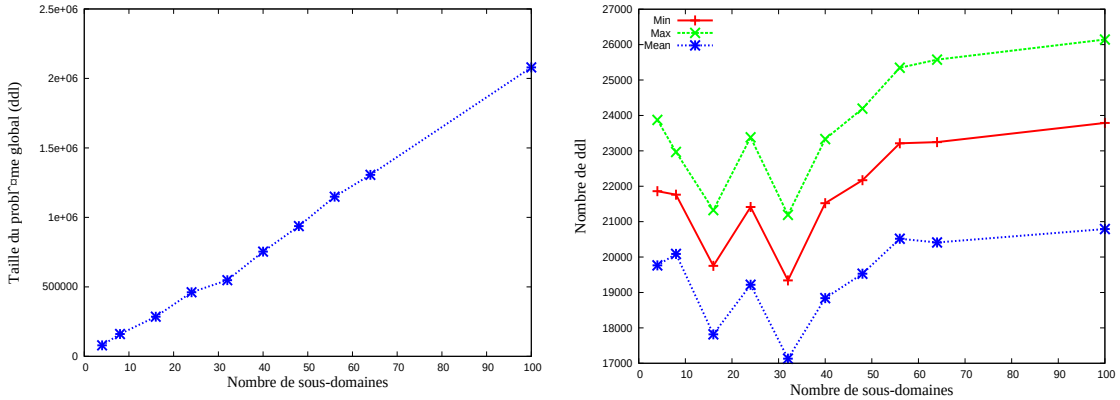


Figure 5 – Taille des problèmes.

Les résultats en parallèle sont présentés sur les figures 6 et 7. Une analyse de ces résultats montre qu'il est très facile d'obtenir de meilleures performances en résolvant des problèmes de plus en plus grands. Lorsque le nombre de sous-domaines augmente la communication devient

aussi importante, ce qui diminue la performance de la méthode FETI. Dans cet exemple, cela signifie que la taille des sous-domaines choisie (environ 20000 inconnues, figure 5(b)) est trop petite. Le temps d'exécution (min ou max) nécessaire pour obtenir les solutions locales est faible comparé au temps de commmunication. Le résultat est clair : nous devons utiliser des sous-domaines de grande taille et une version parallèle du solveur Dissection pour pouvoir atteindre des performances maximales. Tous ces tests sont effectués avec la version du solveur sans multi-threading.

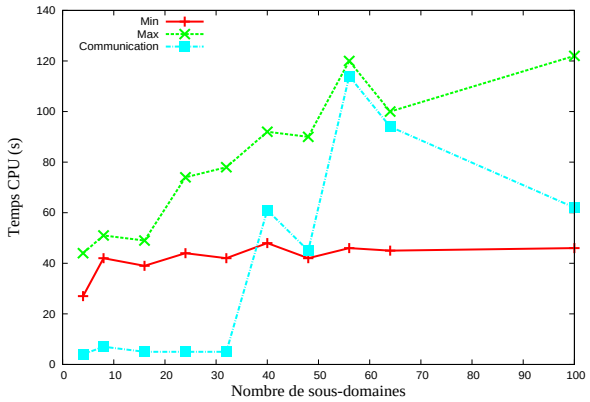


Figure 6 – Temps d'exécution du solveur et communication.

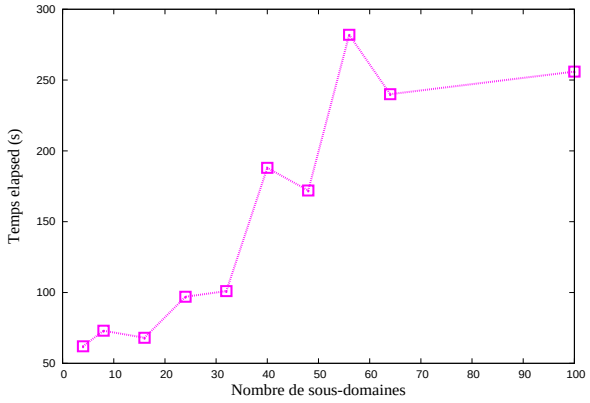


Figure 7 – Temps total exécution.

4 Conclusion

Nous avons mis en place un solveur direct parallèle basé sur la technologie du multi-threading et utilisant un algorithme de dissection emboîtée qui, grâce à sa stratégie de "division et conquête", a conduit à un parallélisme de haut niveau. Ce solveur a été intégré dans les méthodes FETI et il est capable de traiter automatiquement et proprement les modes à énergie nulle dans les sous-structures flottantes. Suite à son implantation dans le code ZéBuLon, de grands problèmes de mécanique ont été simulés en parallèle et nous avons obtenu de bonnes performances.

L'exploitation du plus haut niveau de parallélisme est en cours de réalisation. Il consiste à améliorer, avec l'aide du solveur direct mis en œuvre, la méthode itérative utilisée pour résoudre le problème d'interface de la méthode FETI.

Références

[1] P. Charrier and J. Roman. Algorithmique et calculs de complexité pour un solveur de type dissections emboîtées. *Numerische Mathematik*, vol. 55, 463–476, 1989.

[2] J. J. Dongarra, I. S. Duff, J. Du Croz and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, vol. 16, 1–17, 1990.

[3] C. Farhat and F.-X. Roux. Implicit parallel processing in structural mechanics. *Computational Mechanics Advances*, vol. 2 n° 1, 1–124, 1994.

[4] C. Farhat, K. Pierson and M. Lesoinne. The second generation FETI methods and their application to the parallel solution of large-scale linear and geometrically non-linear structural analysis problems. *Computer Methods in Applied Mechanics and Engineering*, vol. 184 n° 2-4, 333–374, 2000.

[5] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, vol. 10 n° 2, 345–363, 1973.

[6] A. George and W.-H. Liu. Computer solution of large sparse positive definite systems. *Prentice Hall*, 1981.

[7] P. Gosselet and C. Rey. Non-overlapping domain decomposition methods in structural mechanics. *Archives of Computational Methods in Engineering*, vol. 13 n° 4, 515–572, 2006.

[8] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graph. *SIAM Journal on Scientific Computing*, vol. 20 n° 1, 359–392, 1999.

[9] P. Raghavan. DSCPACK home page. *http ://www.cse.psu.edu/ raghavan/Dscpack*, 2001.

[10] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, vol. 55 n° 11, 1801–1809, 1967.